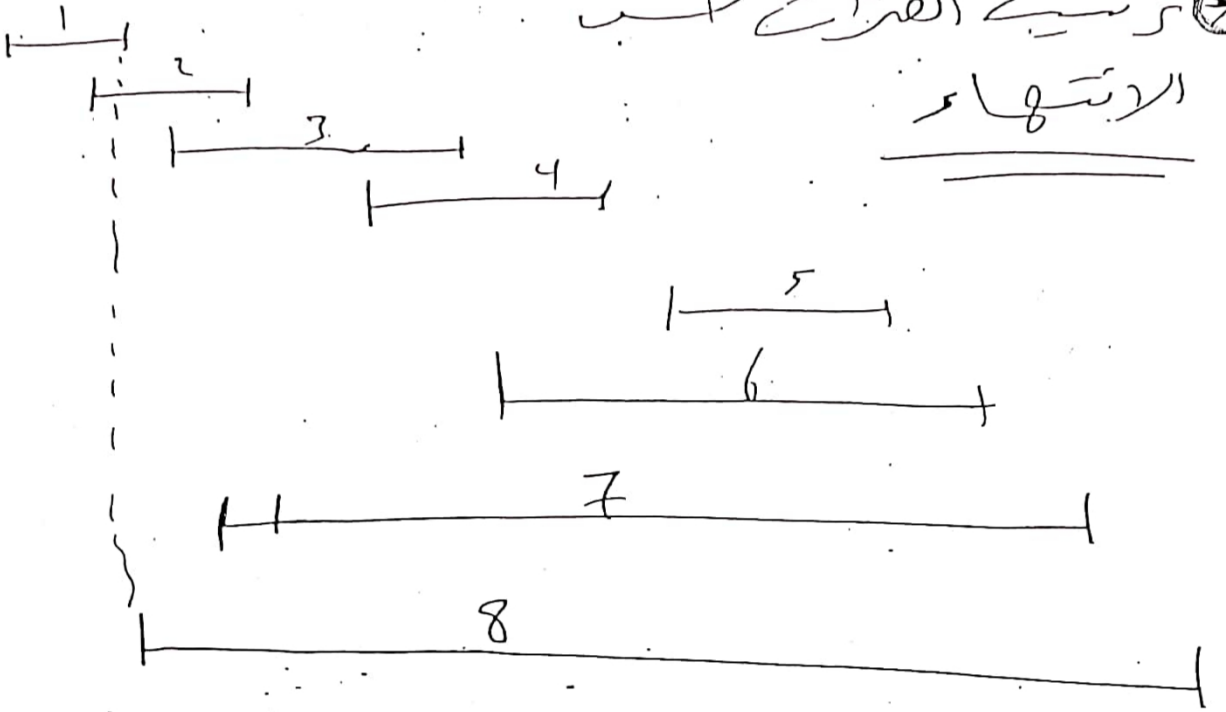
(9) Example

* ترتيب الفترات حسب الانتهاء

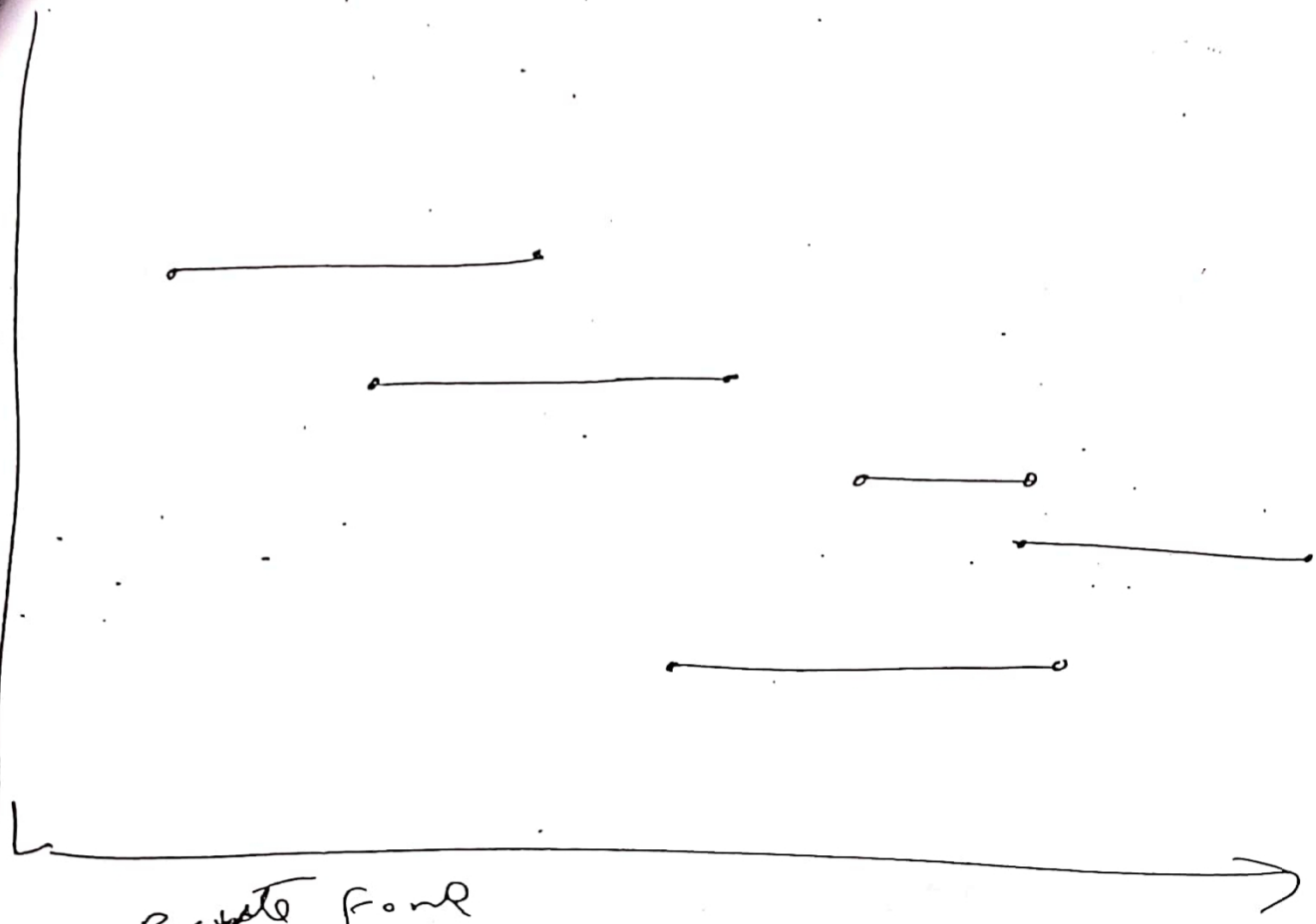


$w_i = i$

i	P(i) →
1	0
2	0
3	1
4	2
5	4
6	3
7	4
8	1

اخر شغل قبل i متوافق مع i

Interval scheduling



Brute force

$$opt(i) = \max \begin{cases} \text{value of opt sol} \\ \text{consists of } 1,2,\ldots i \end{cases} \{ w_i + opt(p(i)), opt(i-1) \}$$

dynamic

$$M[i] = \max \begin{cases} w_i + M[p(i)] \\ M[i-1] \end{cases}$$

```
for(i = 1 to n){

}
```

الحل الفعلي يتم استخدام حسب

ال Algorithm التالي.

```
Fs(i)
    if (i == 0)  output φ
    else {
        if (w_i + M[P_i] ≥ M[i-1])
            output i        calc P(i)
                            قيمة i جاهزة
        else
            output Fs(i-1)
```

(10)

M

0	1	2	3	4	5	6	7	8
0	1	2	4	6	11	11	11	11

$i = 1$

$$\max \begin{cases} w_1 + M[p(1)] \\ = 1 + M[0] = 1 + 0 = 1 \end{cases}$$

$M[0] = 0$

$i = 2$

$$\max \begin{cases} w_2 + M[p(2)] \\ = 2 + M[0] = 2 + 0 = 2 \end{cases}$$

$M[1] = 1$

$i = 3$

$$\max \begin{cases} w_3 + M[p(3)] \\ 3 + M[1] = 3 + 1 = 4 \end{cases}$$

$M[2] = 2$

$i = 4$

$$\max \begin{cases} w_4 + M[p(4)] \\ 4 + M[2] = 4 + 2 = 6 \end{cases}$$

$M[3] = 4$

المجموعات
Set ←

مطلوب ايجاد ... optimal solution

الى انتجيت back tracking

$i = 8$

$w_8 + M[P(8)] \geq M[7]$?

$8 + 1 \geq 11$? no

$i = 7$ $\quad 7 + M[P(7)] \geq M[6]$?

$7 + 1 \geq 11$? no

$i = 6$ $\quad 6 + 4 \geq M[5]$? no

$i = 5$ $\quad 5 + 6 \geq M[4]$? yes

$i = (5)$ is in solution

نبدأ لفتة جديدة

$i = P(5) = 4$

i = 5

$$\max \begin{cases} w_5 + M[p(5)] \\ = 5 + M[4] = 5 + 6 = 11 \\ M[4] = 6 \end{cases}$$

i = 6

$$\max \begin{cases} 6 + 4 = 10 \\ (11) \end{cases}$$

i = 7

$$\max \begin{cases} w_7 + M[p(7)] = 7 + M[1] = 8 \\ M[6] = (11) \end{cases}$$

i = 8

$$\max \begin{cases} w_8 + M[p(8)] = 8 + 1 = 9 \\ M[7] \leq (11) \end{cases}$$

optimal solution = (11)

$4 + 2 \geq 4$ yes

$i = 4$ in solution

$i = p(4) = 2$

$2 + 0 \geq 1$ yes

$i = 2$ in solution

$i = p(2) = 0$

Set = $\{2, 4, 5\}$

Optimal = 11 ← weights کی جمع

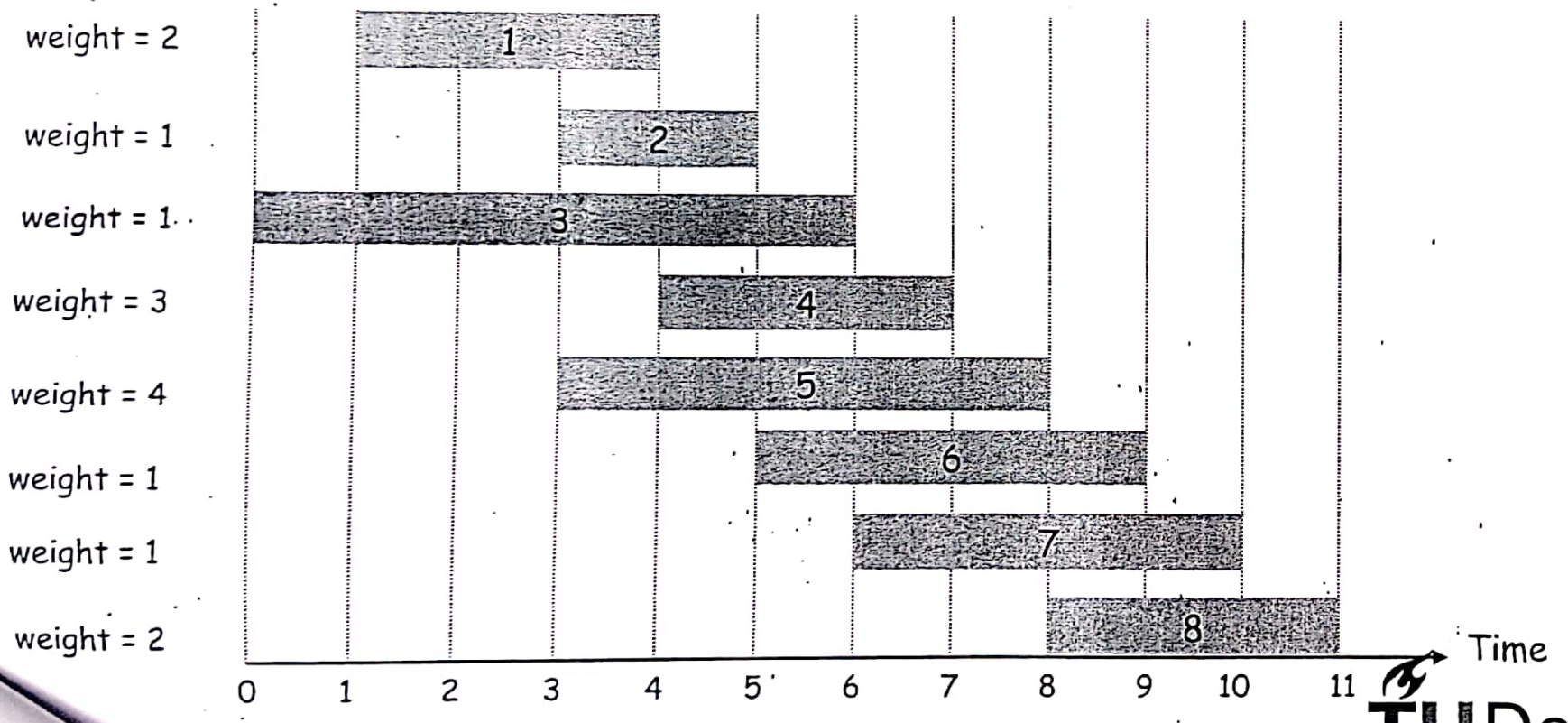
Weighted Interval Scheduling

Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq \ldots \leq f_n$.

Def. p(j) = largest index i < j such that job i is compatible with j. (predecessor)

Q. p(8) = ?, p(7) = ?, p(2) = ?.

A. p(8) = 5, p(7) = 3, p(2) = 0.



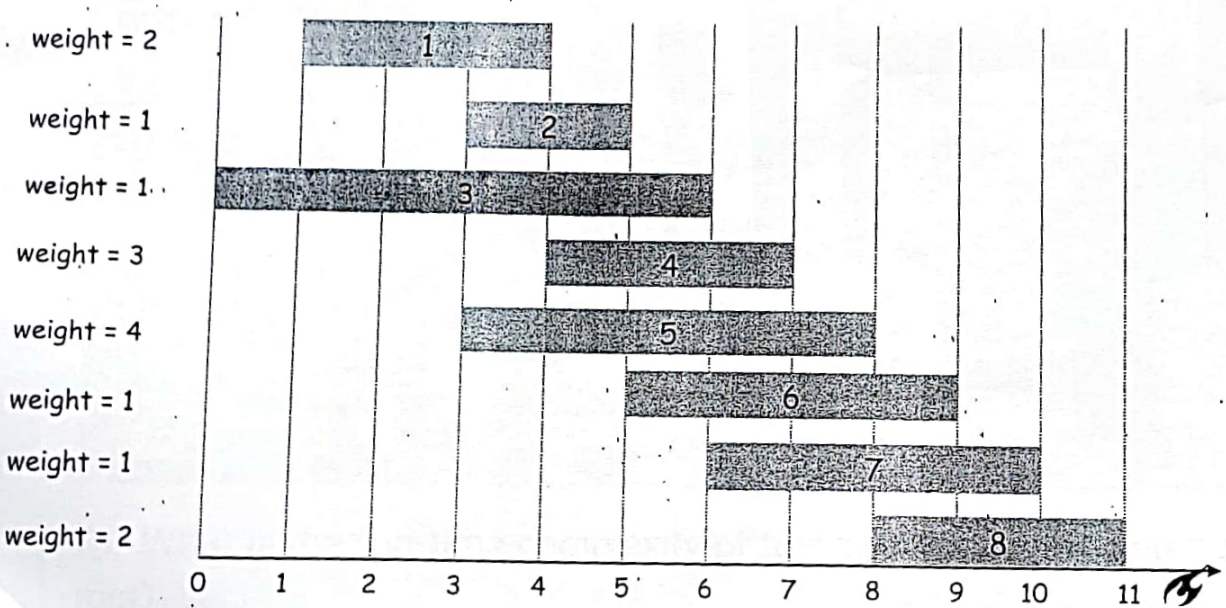
Weighted Interval Scheduling

Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq \ldots \leq f_n$.

Def. p(j) = largest index i < j such that job i is compatible with j. (predecessor)

Q. p(8) = ?, p(7) = ?, p(2) = ?.

A. p(8) = 5, p(7) = 3, p(2) = 0.



Weighted Interval Scheduling: Memoization

Memoization. Store results of each sub-problem in a cache; lookup as needed.

```
Input: n, s1,…,sn, f1,…,fn, v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ … ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n
    M[j] = empty   ← global array
M[0] = 0

M-Compute-Opt(j) {
    if (M[j] is empty)
        M[j] = max(vj + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
    return M[j]
}
```

Q. What is the run-time complexity of this algorithm with memoization? (1 min)

Weighted Interval Scheduling: Memoization

Memoization. Store results of each sub-problem in a cache; lookup as needed.

```
Input: n, s1,…,sn, f1,…,fn, v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ … ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n
    M[j] = empty   ← global array
M[0] = 0

M-Compute-Opt(j) {
    if (M[j] is empty)
        M[j] = max(vj + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
    return M[j]
}
```

Q. What is the run-time complexity of this algorithm with memoization? (1 min)

TU Delft

Weighted Interval Scheduling: Running Time

Claim. Memoized version of algorithm takes O(n log n) time.

Proof.

Q. How many iterations in initialization?

Sort by finish time: O(n log n).

Computing p(·) : O(n) by decreasing start time

Q. How many iterations in one invocation?

`M-Compute-Opt(j)`: each invocation takes O(1) time and either

- (i) returns an existing value `M[j]`
- (ii) fills in one new entry `M[j]` and makes two recursive calls

Q. How many invocations?

Progress measure Φ = # nonempty entries of `M[]`.

- initially $\Phi = 0$, throughout $\Phi \leq n$.
- (ii) increases Φ by 1 and only then at most 2 recursive calls.

Overall running time (without init) of `M-Compute-Opt(n)` is O(n). ▪

Remark. O(n) if jobs are pre-sorted by start and finish times.

Weighted Interval Scheduling: Running Time

Claim. Memoized version of algorithm takes O(n log n) time.

Proof.

Q. How many iterations in initialization?

Sort by finish time: O(n log n).

Computing p(·): O(n) by decreasing start time

Q. How many iterations in one invocation?

`M-Compute-Opt(j)`: each invocation takes O(1) time and either

- (i) returns an existing value `M[j]`
- (ii) fills in one new entry `M[j]` and makes two recursive calls

Q. How many invocations?

Progress measure Φ = # nonempty entries of `M[]`.

- initially $\Phi = 0$, throughout $\Phi \leq n$.
- (ii) increases Φ by 1 and only then at most 2 recursive calls.

Overall running time (without init) of `M-Compute-Opt(n)` is O(n). ▪

Remark. O(n) if jobs are pre-sorted by start and finish times.

Problem 1

Solve the following instance of the Knapsack problem using dynamic programming paradigm. The maximum allowed weight is Wmax = 10.

i	1	2	3	4
Vi	30	20	25	41
Wi	5	1	5	4

a- Give the recursive equation you used to define the data structure needed for your dynamic programming solution. Then, fill the proposed data structure.

b- What is your solution to this instance of the Knapsack problem?

Problem 2

Suppose X= peas and Y= teams.

(a) Compute the length of an LCS of X and Y by filling out the c-matrix below

		t	e	a	m	s
p						
e						
a						
s						

(b) What is LCS of X and Y? For Xi and Yj to be match in the LCS, what must be true about c[i,j]?

if(X[i] == Y[j])

c[i,j] = c[i-1,j-1]+1

Problem 3

Our college is holding a conference on advances in computer algorithms. The conference lasts one day only (from 08.00 am to 20.00). The conference consists of n presentations given by international speakers. Each presentation P is defined by two times **P.start** and **P.end** (**P.start** < **P.end**). In order to encourage you to attend

these presentations, your CSC 311 decided to give you **5 points** for each attended presentation regardless of its duration. Student cannot attend two presentations at the same time, and must stay till the end of the presentation to get the bonus.

Assume that the presentation information are available in an array **P**, i.e., **P[i].start** and **P[i].end** are the start and the finish time of the i^{th} presentation, respectively.

a- Describe an algorithm that assists the student to maximize his bonus credit. The algorithm should print the **.start** and **.end** properties of presentations to which he should attend.

b- What is the time complexity of your algorithm? Which programming design technique have you used?

LCS recursive solution

- We start with $i = j = 0$ (empty substrings of x and y)
- Since X_0 and Y_0 are empty strings, their LCS is always empty (i.e. $c[0,0] = 0$)
- LCS of empty string and any other string is empty, so for every i and j: $c[0, j] = c[i,0] = 0$

LCS recursive solution

- When we calculate *c[i,j]*, we consider two cases:
- **First case:** *x[i]=y[j]*: one more symbol in strings X and Y matches, so the length of LCS X_i and Y_j equals to the length of LCS of smaller strings X_{i-1} and Y_{i-1} , plus 1

5/

LCS recursive solution

- **Second case:** *x[i] != y[j]*

As symbols don't match, our solution is not improved, and the length of $LCS(X_i , Y_j)$ is the same as before (i.e. maximum of $LCS(X_i, Y_{j-1})$ and $LCS(X_{i-1}, Y_j)$

Why not just take the length of $LCS(X_{i-1}, Y_{j-1})$?

LCS Length Algorithm

```
LCS-Length(X, Y)
1. m = length(X)  // get the # of symbols in X
2. n = length(Y) // get the # of symbols in Y
3. for i = 1 to m        c[i,0] = 0      // special case: Y0
4. for j = 1 to n        c[0,j] = 0      // special case: X0
5. for i = 1 to m                        // for all Xi
6. for j = 1 to n                        // for all Yj
7.       if ( Xi == Yj )
8.              c[i,j] = c[i-1,j-1] + 1
9.       else c[i,j] = max( c[i-1,j], c[i,j-1] )
10. return c
```

عندما يتساوون

عندما يختلفون

Problem 7

@ (1) - sort All presentations by P[i].end → O(n log n)

for i = 1 → n {

(2) - let p(i) = j such that j is last or greatest j satisfy P[j].end ≤ P[i].start → O(n)

}

M[0] = 0

(3) for i = 1 to n

```
if( 5 + M[p(i)] > M[i-1] )
    M[i] = 5 + M[p(i)];
else
    M[i] = M[i-1];
```

→ O(n)

return M[n];

Complexity is depending on sorting algorithm / n log n

for → O(n)

∴ Total O(n log n)

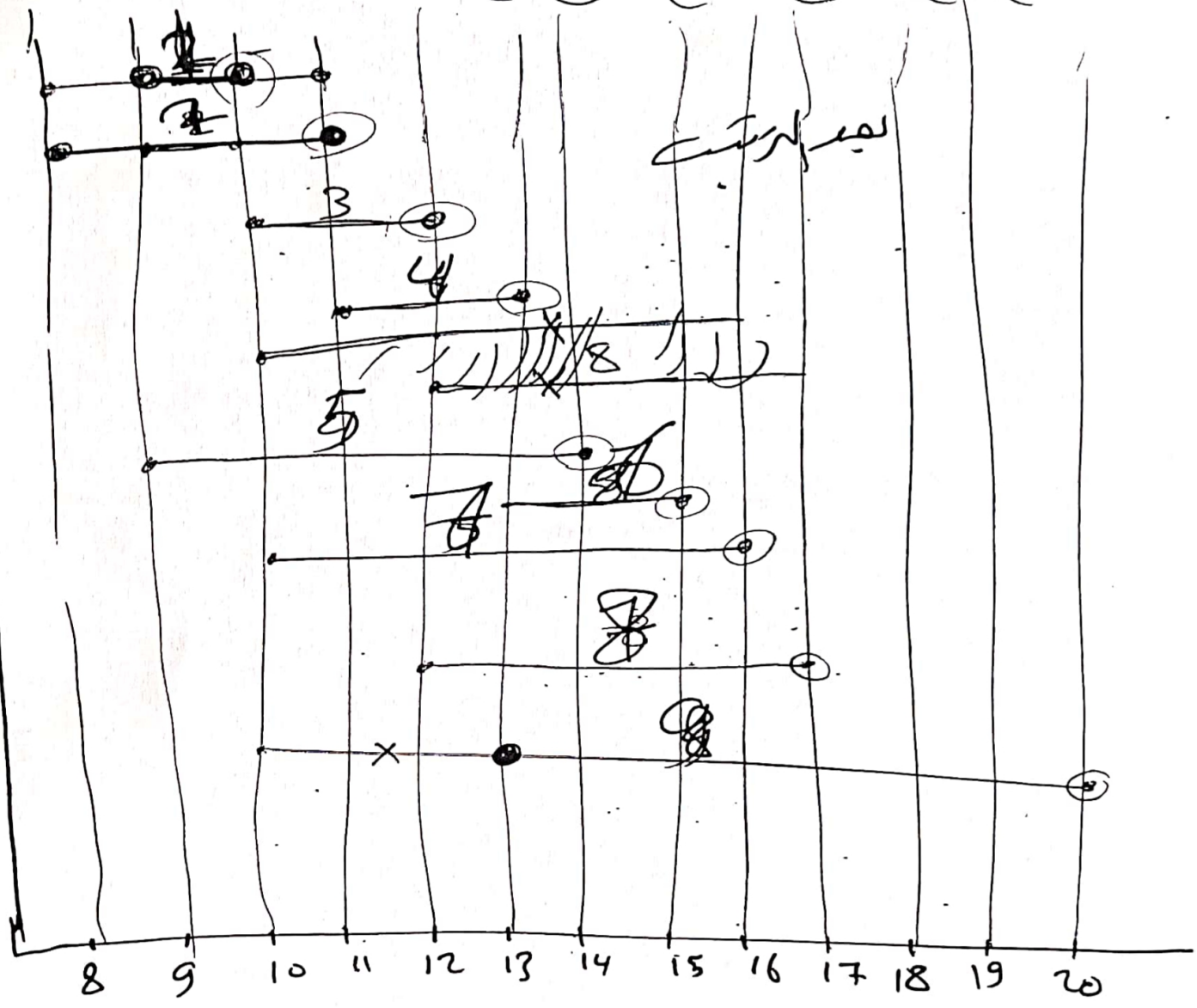
(8)

For printing the solution

```
FS(i)
    if(i==0)   output ϕ
    else if(5+M[Pi] >= M[i-1])
        output i;
    output FS(P(i))
    else
        output FS(i-1)
```

Example 9

i	1	2	3	4	5	6	7	8	9
P[i].st	8	9	10	9	11	11	12	13	13
P[i].en	11	10	12	14	18	13	17	15	20
ترتيب	2	1	3	5	7	4	8	6	9



i	1	2	3	4	5	6	7	8	9
r(i)	0	0	1	2	0	4	1	3	4

0	1	2	3	4	5	6	7	8	9	10
0	5	5	10	10	10	15	15	15	15	

$i = 1$

$$\max \begin{cases} 5 + M[p(1)] = 5 + 0 = 5 \\ 0 \end{cases}$$

($M[0]$)

$i = 2$

$$\max \begin{cases} 5 + M[p(2)] = 5 + 0 = 5 \\ 5 \end{cases}$$

$i = 3$

$$\max \begin{cases} 5 + M[r(3)] = 5 + 5 = 10 \\ 5 \end{cases}$$

$i = 4$

$$\max \begin{cases} 5 + M[r(4)] = 5 + M[2] = 10 \\ 10 \end{cases}$$

$i = 5$

$$\max \begin{cases} 5 + M[r(5)] = 5 + 0 = 5 \\ 10 \end{cases}$$

$i = 6$

$$\max \begin{cases} 5 + M[r(6)] = 5 + M[4] = 15 \\ 10 \end{cases}$$

$i = 7$

$$\max \begin{cases} 5 + M[r(7)] = 5 + 5 = 10 \\ 15 \end{cases}$$

$i = 8$

$$\max \begin{cases} 5 + M[r(8)] = 5 + 10 = 15 \\ 15 \end{cases}$$

$i = 9$

$$\max \begin{cases} 5 + M[r(9)] = 5 + 10 = 15 \\ 15 \end{cases}$$

solution = {1, 3, 6}

(11)

Let us say that you are given a number N you have to find the number of different ways to write it as the sum of 1, 3, and 4

Example Let $N = 5$

$\therefore \quad 1+1+1+1+1 = 5$

$1+4 = 5$

$1+1+3 = 5$

$4+1 = 5$

$3+1+1 = 5$

$1+3+1 = 5$

6 ways $\quad \therefore DP_5 = 6$ ways

$DP_4 = ?? = 3$

$DP_3 = ?? = 1$

$DP_1 = ?? = 1$

$1+1+1+1 = 4$
$1+3 = 4$
$3+1 = 4$
4 ~~$= 4+4$~~ } 3 ways

$1+1+1 = 3$
$3 = 3$

$DP_5 = DP_4 + DP_2 + DP_1 = 4 + 1 + 1 = 6$ ways

Algorithm:- (12)

DP[0] = DP[1] = DP[2] = 1 , DP[3] = 2

for (i = 4, i <= n; i++) {

DP[i] = DP[i-1] + DP[i-3] + PP[i-4].

}

For n = 5 DP(4) مثال

1+1+1+1+1 + ✗ = 5

DP(1)

4 + 4 = 5

DP(2)

4 + 1 + 3 = 5

DP(4)

4 + ✗ = 5

DP(4)

3 + 1 + ✗ = 5

1 + 3 + ✗ = 5

DP(4)